CSC584 Homework 2
Student: Tzu Tang(Leslie) Liu
Unity ID: tliu33

Part1. Variable Matching Steering Behaviours

In my implementation, I use a Kinematic structure to store a character's position, orientation, velocity, and rotation, which is shared between the character and the target (the mouse). A virtual class called SteeringBehavior defines a method, VariableMatch(), to align the character's kinematic state with the target's. I implemented four subclasses for matching position, orientation, velocity, and rotation, but for Part 1, the focus is on VelocityMatchingBehavior.

The Character class, represented as a boid image, maintains its own kinematic state. Each frame, the character updates its velocity to match the mouse's and then moves using a frame-rate–independent update. Boundary conditions are handled with a wrap-around method: if the sprite exits one side of the screen, it reappears on the opposite side.

This method produces very responsive motion since the character's velocity directly follows the mouse's. However, the behavior is sensitive to noise in mouse movement, causing noticeable oscillations. In future work, I plan to explore blending multiple behaviors and applying input smoothing techniques.

Part2. Arrive and Align

In my implementation, to get the wander feature, I extended the basic variable-matching steering framework to include two specific behaviors: Arrive and Align. And guide a character toward a target point set by a mouse click, ensuring smooth deceleration as it approaches and a gradual rotation to face its direction of motion.

I experimented with two different parameter sets for arrival—one that begins deceleration earlier with a lower top speed, and another with later deceleration and a higher top speed. The first parameter set can make a natural and smooth slowing down, reducing the likelihood of overshooting the target, and lower top speed allows for more precise positioning when arriving at the target, but may take longer travel time and feel sluggish when trying to quickly respond to a sudden change of target. The second parameter set can cover larger distances quickly, making it more reactive to target changes. While the transition from high speed to stopping might feel less smooth if not carefully tuned, potentially leading to a jerky arrival.

Compare the drawback of these two parameter sets. In the case of drawing a small circle, I believe the first parameter set is more suitable.

Switching between these methods (via the spacebar) revealed that subtle changes in the slow radius and maximum speed dramatically affect the smoothness and responsiveness of arrival.

AlignBehavior class is responsible for smoothly rotating the character so that its orientation matches the direction of motion. Based on the character's current velocity, the desired orientation is computed using the arctangent function.

The angle difference between the desired and current orientations is then calculated and normalized to a range of –180° to 180°. If the difference is small enough, the character is snapped directly to the desired angle. Otherwise, the character rotates at a speed scaled by the magnitude of the angle difference. This ensures that the rotation is smooth.

Breadcrumb is implemented by a double-ended queue. Which is recorded every few frames. I maintain a fixed number of breadcrumbs, so older points are removed as new ones are added.

While the current design works well for a single target point, If I had the chance to better improve the motion of the sprite, I'd like blending additional behaviors (e.g., obstacle avoidance) or incorporating input smoothing techniques for more robust target detection.

Part3. Wander

I tried two different methods for changing orientation, for method 1 I apply Circle Wander which calculates a projected circle in front of the character. A small wander rate means the displacement on this circle changes slowly, which results in a relatively smooth path changing.

Method 2 I use Jitter Wander which applies a direct random jitter to the orientation. Make a more erratic rotation depending on the same parameter values. In my code I can switch between these two methods via the spacebar.

If I had the chance to better improve the motion of the sprite, I'd like to explore the dynamic parameter adjustment during the wander process. Instead of a fixed wander rate, adjust it dynamically based on the sprite's speed or environmental factors. For instance, slowing down the wander rate when the sprite nears obstacles can prevent abrupt turns. Another thing I'd like to explore is to integrate wander behavior with other steering behaviors such as seeking, fleeing, or obstacle avoidance. By blending these forces, you can achieve more complex and realistic movement patterns.

Part4. Flocking

My flocking algorithm combines three steering forces: Separation, Alignment, and Cohesion. In my implementation, each boid is represented by its own class, which stores its kinematic properties and a breadcrumb trail that visually traces its path. During my experiment for adjust the weight of each force, I noticed that the Separation force makes a boid steer away from neighbors that are too close by calculating a repulsive vector that grows stronger as the distance decreases. The Alignment force guides a boid to match its neighbors' velocities by averaging their speeds and directions, while the Cohesion force pulls a boid toward the center of nearby flock mates by moving it toward the average position of its neighbors.

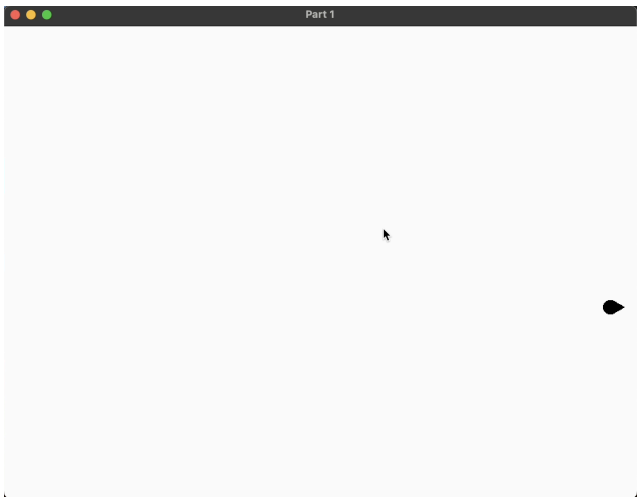
Each boid's total steering force is the weighted sum of these three components, with the weights easily adjusted to fine-tune the behavior. Speed and steering force limits ensure that movement remains smooth and realistic.

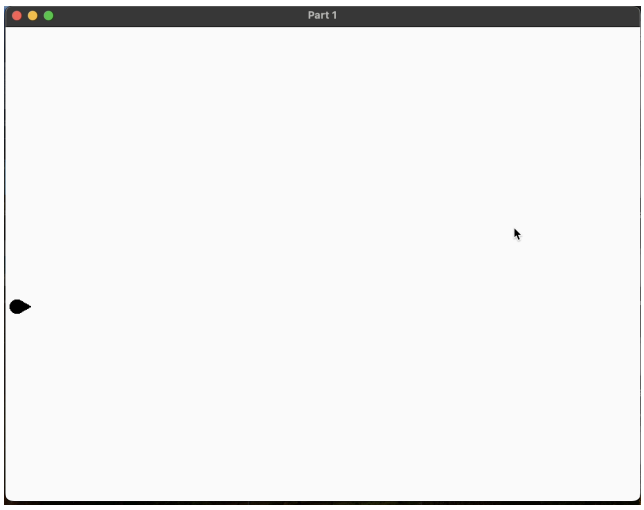
A boundary check mechanism handles boundaries, so boids reappear on the opposite side of the screen if they leave the visible area. The use of a semi-transparent green breadcrumb trail is also used for the visualization, providing insight into each boid's recent movements.

Tzu Tang(Leslie) Liu (tliu33@ncsu.edu)

APPENDIX

Part1:

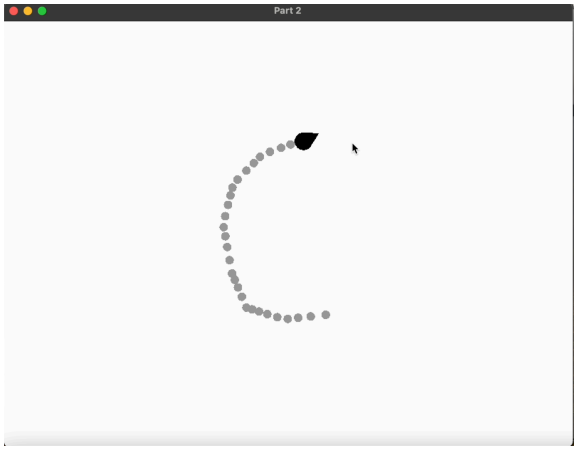




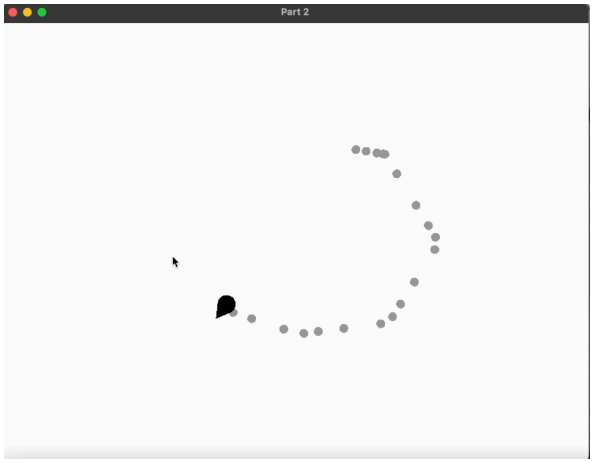
Part2:

Drawing circle by Arrival with begins deceleration earlier with a lower top speed

Tzu Tang(Leslie) Liu (tliu33@ncsu.edu)

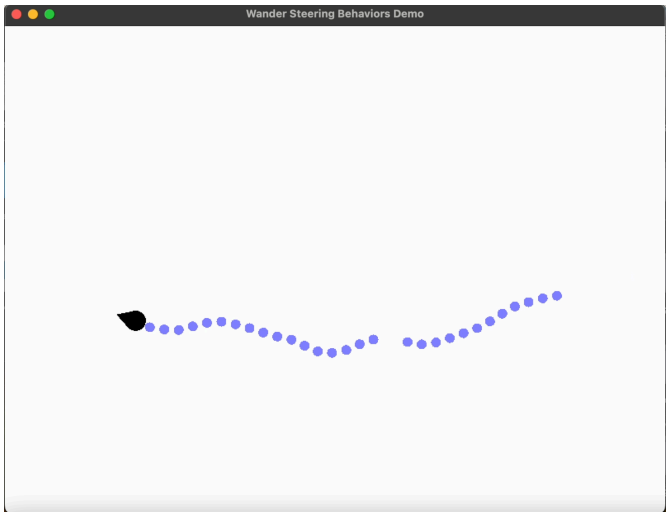


Drawing circle by Arrival with later deceleration and a higher top speed



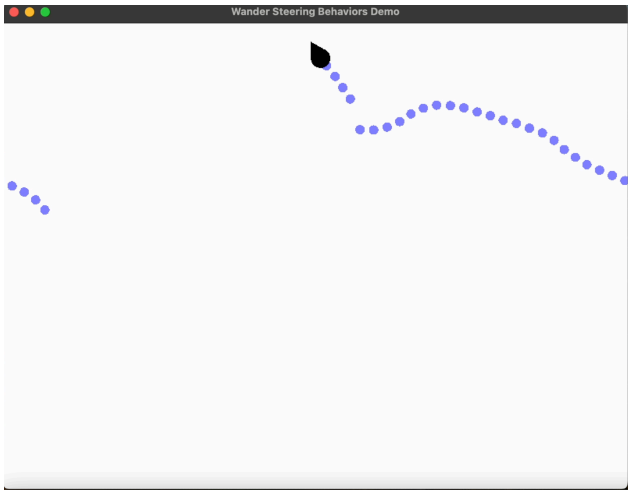
Part3:

Method 1 (Circle Wander):

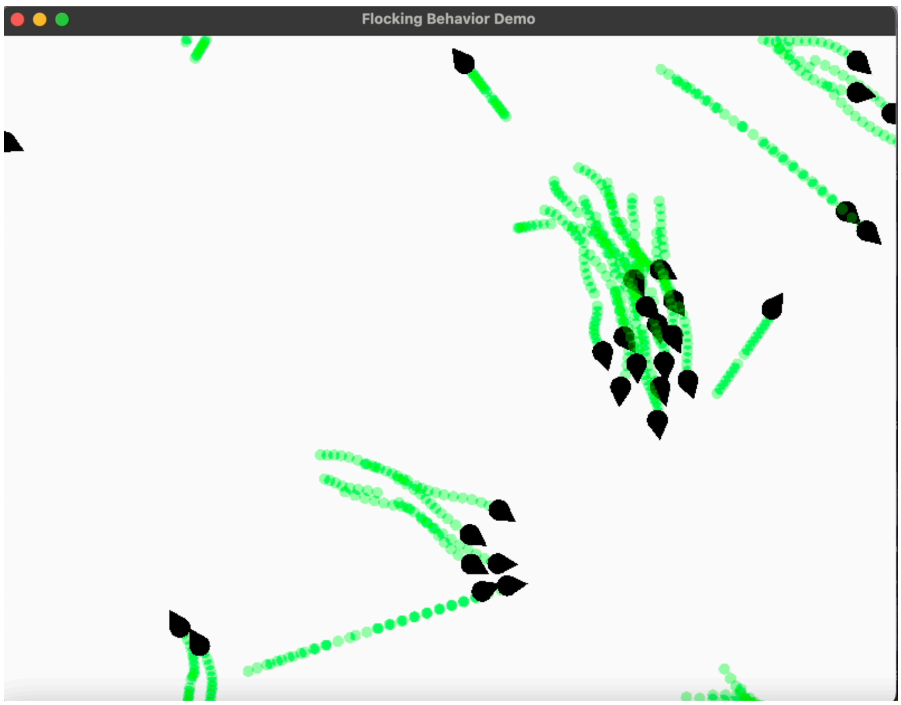


Tzu Tang(Leslie) Liu (tliu33@ncsu.edu)

Method 2 (Jitter Wander):



Part4:



Tzu Tang(Leslie) Liu (tliu33@ncsu.edu)

Tzu Tang(Leslie) Liu (tliu33@ncsu.edu)